Readers and Writers SE 3BB4

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Э

▶ < ≣ ▶ ...

Readers and Writers

- A shared database is accessed by two kinds of processes.
 Readers execute transactions that examine the database while Writers both examine and update the database. A Writer must have exclusive access to the database; any number of Readers may concurrently access it.
- Events or actions of interest: acquireRead, releaseRead, acquireWrite, releaseWrite
- Processes: Readers, Writers, RW_Lock
- Properties: *RW_Safe*, *RW_Progress*



```
set Actions =
{acquireRead,releaseRead,acquireWrite,releaseWrite}
READER = (acquireRead->examine->releaseRead->READER)
+ Actions
\ {examine}.
WRITER = (acquireWrite->modify->releaseWrite->WRITER)
+ Actions
\ {modify}.
```

Alphabet extension is used to ensure that the other access actions cannot occur freely for any prefixed instance of the process (as before).

Action hiding is used as actions **examine** and **modify** are not relevant for access synchronisation.

・ロト ・回ト ・ヨト ・ヨト … ヨ

```
The lock
const False = 0 const True = 1
                                             maintains a count
range Bool = False..True
const Nread = 2 // Maximum readers
                                             of the number of
                       // Maximum writers
                                             readers, and a
const Nwrite= 2
                                             Boolean for the
RW LOCK = RW[0][False],
                                             writers.
RW[readers:0..Nread][writing:Bool] =
     (when (!writing)
          acquireRead -> RW[readers+1][writing]
     /releaseRead -> RW[readers-1][writing]
     when (readers==0 && !writing)
          acquireWrite -> RW[readers][True]
     /releaseWrite -> RW[readers][False]
     ).
```

<ロ> <四> <四> <四> <四> <四> <四</p>

Safety Property

```
property SAFE_RW
= (acquireRead -> READING[1]
    |acquireWrite -> WRITING
    ),
READING[i:1..Nread]
= (acquireRead -> READING[i+1]
    |when(i>1) releaseRead -> READING[i-1]
    |when(i=1) releaseRead -> SAFE_RW
    ),
WRITING = (releaseWrite -> SAFE_RW).
```

We can check that **RW_LOCK** satisfies the safety property.....

||READWRITELOCK = (RW_LOCK || SAFE_RW).

 Note that we do not check this property for the whole system, only for one component namely *RW*_LOCK. This is computationally simpler.

Explicit Safety Property for 2 Readers and 2 Writers



- An *ERROR* occurs if a reader or writer is badly behaved (*release* before *acquire* or more than two readers).
- However when composing with *READWRITELOCK* such bad behaviour is not allowed.





- Neither deadlock nor safety violation.
- It requires a tool to show it, the tool is not efficient (it cannot be).
- Try the tool for 10 readers and 10 writers!
- It is always better if some properties can just be **proved**, **not only checked**.
- Problem with checking: one cannot checked the case of *n* readers and *m* writers, only, say, 5 readers and 4 writers, etc.

▶ ★ 臣 ▶ ...

progress WRITE = {writer[1..Nwrite].acquireWrite}
progress READ = {reader[1..Nread].acquireRead}

WRITE - eventually one of the writers will acquireWrite

- **READ** eventually one of the readers will acquireRead
- I do not like it! Why not all?
- Actually this problem shows well the limits of pure FSP model.

- No FAIR CHOICE assumption: both write and read can starve.
- FAIR CHOICE assumption: both write and read are live.
- But waht in *real world* th assumption of *FAIR CHOICE* mean?
- This is assumption about a *possibility* of *clever implementation*.

- Simple use of priorities does not guarantee liveness.
- We lower the priority of the release actions for both *readers* and *writers*.

||RW_PROGRESS = READERS_WRITERS
>>{reader[1..Nread].releaseRead,
writer[1..Nwrite].releaseWrite}.

- Progress violation: WRITE
- Path to terminal set of states: reader.1.acquireRead
- Actions in terminal set:

 $\{\textit{reader.1.acquireRead},\textit{reader.1.releaseRead},$

reader.2.acquireRead, reader.2.releaseRead}

• *WRITER* starvation: the number of readers never drops to zero!



WRITER Priority

• Block readers if there is a writer waiting.

```
RW_LOCK = RW[0][False][0],
RW[readers:0..Nread][writing:Bool][waitingW:0..Nwrite]
= (when (!writing && waitingW==0)
acquireRead -> RW[readers+1][writing][waitingW]
|releaseRead -> RW[readers+1][writing][waitingW]
|when (readers==0 && !writing)
acquireWrite-> RW[readers][True][waitingW-1]
|releaseWrite-> RW[readers][False][waitingW]
|requestWrite-> RW[readers][waitingW]
|requestWrite-> RW[readers][writing][waitingW+1]
).
```

property RW SAFE:

No deadlocks/errors

progress READ and WRITE:

```
Progress violation: READ
Path to terminal set of states:
    writer.1.requestWrite
Actions in terminal set:
{writer.1.requestWrite, writer.1.acquireWrite,
writer.2.acquireWrite, writer.2.requestWrite,
writer.2.acquireWrite, writer.2.requestWrite}
```

In practice, this may be satisfactory as is usually more read access than write, and readers generally want the most up to date information.

11/38

イロト (四) (注) (こ) (こ)

- We have encountered many problems both with formulation of the problem and solutions to it.
- Let us try another formalism.



< ∃⇒

 We have *n* processes, *n* > 0, which may read and write in a shared memory. Several precesses may be reading concurrently, but when a process is writing, no other process can be reading or writing. No priority is assumed.

Readers and Writers: P/T Net Model



∢ 臣 ≯

E.

Place/Transitions Nets (P/T-Nets)

• Firing rules:



• Different kind of simultaneity:



臣

Incidence Matrix

• P - places, T - transitions









Multisets (Bags, Weighted Sets)

- A multiset *m*, over a non-empty and finite set *S* is a function $m: S \rightarrow \mathbb{N} = \{0, 1, 2, ...\}$
- m(s) is the number of appearances of s in m.
- notation: *M* is usually represented by:

$$\sum_{s\in S} m(s)s$$

$$S = \{a, b, c, d, e\},$$

 $m(a) = 3, m(b) = 1, m(c) = 0, m(d) = 183, m(e) = 4$
 $m = 3a + b + 4e + 183d$

- $s \in m \iff m(s) \neq 0$
- m(s) is a coefficient
- the empty multiset $m = \emptyset \iff m(s)$ for each $s \in S$.

Basic Definitions

- Let x be a multiset (weighted set) of transitions, i.e. $\mathbf{x}: \mathcal{T} \to \mathbb{N}$
- **x** is **positive** iff $\mathbf{x}(t) > 0$ for at least one $t \in T$, i.e. $\mathbf{x} \neq \emptyset$
- Marking: m : P → N.
 Marking is not interpreted as a multiset!
- $m \ge m' \iff \forall p \in P. \ m(p) \ge m'(p).$
- Assumption: Each place can hold an arbitrary number of tokens.
- Let W^- be the following matrix:

$$orall (p,t)\in P imes T. \ W^-(p,t)= \left\{egin{array}{cc} -W(p,t) & ext{if } W(p,t)<0 \ 0 & ext{if } W(p,t)\geq 0 \end{array}
ight.$$

A positive multiset of transitions x has concession in a marking m iff m ≥ W⁻ · x

↑

matrix multiplication

Example (n = 15)

• $\mathbf{x} = 10t_1 + 3t_2$ has a concession in $m_0 = (15, 0, 0, 0, 0, 15)$, since

$$W^{-} \cdot \mathbf{x} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 15 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = (13, 0, 0, 0, 0, 0),$$

and $m_0 > (13, 0, 0, 0, 0, 0)$.

• $\mathbf{x} = t_4$ does not have a concession in m = (8, 3, 1, 2, 0, 13), since

$$W^{-} \cdot \mathbf{x} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 15 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = (0, 0, 1, 0, 0, 15),$$
and *m* and $(0, 0, 1, 0, 0, 15)$ are incomparable.

- When x has concession, it may fire.
- If x fires, w new marking:

 $m' = m + W \cdot \mathbf{x}$

is reached.

- m' is said to be **directly reachable** from m , i.e. $m \Rightarrow m'$
- $\Rightarrow^* = \bigcup_{i=0}^{\infty} \Rightarrow^i$, or \Rightarrow^* is a reflexive and transitive closure of \Rightarrow , is called **reachability**.

Example

Let $m_0 = (15, 0, 0, 0, 0, 15)$ and $\mathbf{x} = 10t_1 + 3t_2$. We calculate $m_1 = m_0 + W \cdot \mathbf{x}$. $m_1 = m_0 + W \cdot \mathbf{x} =$ $\begin{pmatrix} -1 & -1 & 0 & 1 & 1 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & -15 & 1 & 15 \\ \end{pmatrix} \cdot \begin{bmatrix} 10 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \end{bmatrix} =$ (15, 0, 0, 0, 0, 15) + (-13, 10, 3, 0, 0, 0) = (2, 10, 3, 0, 0, 15).

Invariants

• Let **v** be a **multiset of places**, i.e. $\mathbf{v} : \mathbf{P} \to \mathbb{N}$.

Note that $m: P \to \mathbb{N}$ and $\mathbf{v}: P \to \mathbb{N}$, but the interpretation is different, marking is not interpreted as a multiset!

Theorem (Lautenbach 1979)

Let **v** be a multiset of places. If $\mathbf{v} \cdot W = 0$ and $m \Rightarrow^* m'$ then

$$v \cdot m' = v \cdot m.$$

Proof.

It suffices to show it for
$$m \Rightarrow m'$$
. $v \cdot m' = v \cdot (m + W \cdot \mathbf{x}) = v \cdot m + v \cdot (W \cdot \mathbf{x}) = v \cdot m + (v \cdot W) \cdot \mathbf{x} = v \cdot m + 0 \cdot \mathbf{x} = v \cdot m$.

Definition

A multiset of places **v** is said to be an **invariant** iff $v \cdot W = 0$.

• Each linear combination of invariants is itself an invariant.

Multiplication of a Vector by an Array



Invariants: Example

Example

Consider $i_1 = (1, 1, 1, 1, 1, 0)$, $i_2 = (0, 0, 0, 1, n, 1)$, $i_3 = (-1, -1, -1, 0, n - 1, 1)$. We show that i_1, i_2 and i_3 are invariants.

$$i_{1} \cdot W = \begin{bmatrix} 1\\1\\1\\1\\1\\0 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & 0 & 1 & 1 & 0\\1 & 0 & -1 & 0 & 0 & 0\\0 & 1 & 0 & -1 & 0 & 0\\0 & 0 & 1 & 0 & -1 & 0\\0 & 0 & 0 & 1 & 0 & -1\\0 & 0 & -1 & -n & 1 & n \end{bmatrix} = (0, 0, 0, 0, 0, 0)$$
$$i_{2} \cdot W = \begin{bmatrix} 0\\0\\0\\1\\n\\1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & 0 & 1 & 1 & 0\\1 & 0 & -1 & 0 & 0 & 0\\0 & 1 & 0 & -1 & 0 & 0\\0 & 0 & 1 & 0 & -1 & 0\\0 & 0 & 0 & 1 & 0 & -1\\0 & 0 & -1 & -n & 1 & n \end{bmatrix} = (0, 0, 0, 0, 0, 0)$$
$$i_{3} = i_{2} - i_{1}.$$

Invariant As an Expression

Definition

An invariant can also be defined as a **formula** obtained from $\mathbf{v} \cdot m_0 = \mathbf{v} \cdot m$, where \mathbf{v} is an invariant, as defined previously, m_0 is the initial marking, and m is a marking variable.

Example

$$i_{1} = (1, 1, 1, 1, 1, 0), m_{0} = (n, 0, 0, 0, 0, n).$$

$$i_{1} \cdot m_{0} = (1, 1, 1, 1, 1, 0) \cdot (n, 0, 0, 0, 0, n) = n$$

$$m = (m(LP), m(WR), m(WW), m(R), m(W), m(S))$$

$$i_{1} \cdot m =$$

$$(1, 1, 1, 1, 1, 0) \cdot (m(LP), m(WR), m(WW), m(R), m(W), m(S)) =$$

$$m(LP) + m(WR) + m(WW) + m(R) + m(W).$$

$$i_{1} \cdot m_{0} = i_{1} \cdot m \Longrightarrow$$

$$m(LP) + m(WR) + m(WW) + m(R) + m(W) = n$$

• The number of processes is an invariant.

Example

$$i_{2} = (0, 0, 0, 1, n, 1), m_{0} = (n, 0, 0, 0, 0, n).$$

$$m = (m(LP), m(WR), m(WW), m(R), m(W), m(S))$$

$$i_{2} \cdot m_{0} = (0, 0, 0, 1, n, 1) \cdot (n, 0, 0, 0, 0, n) = n$$

$$i_{2} \cdot m =$$

$$(0, 0, 0, 1, n, 1) \cdot (m(LP), m(WR), m(WW), m(R), m(W), m(S)) =$$

$$m(R) + n \cdot m(W) + m(S).$$

 $i_2 \cdot m_0 = i_2 \cdot m \implies m(R) + n \cdot m(W) + m(S) = n$

- When a process is writing, no other process can be reading or writing.
- The number of reading processes is between 0 and *n*.
- If no process is reading and writing, m(S) = n.
- t₃ has concession if at least one process is waiting to read.
- *t*₄ has concession if at least one process is waiting to write.

Example

$$i_{3} = (-1, -1, -1, 0, n - 1, 1), m_{0} = (n, 0, 0, 0, 0, n).$$

$$m = (m(LP), m(WR), m(WW), m(R), m(W), m(S))$$

$$i_{3} \cdot m_{0} = (-1, -1, -1, 0, n - 1, 1) \cdot (n, 0, 0, 0, 0, n) = 0$$

$$i_{3} \cdot m = (-1, -1, -1, 0, n - 1, 1) \cdot (m(LP), m(WR), m(WW), m(R), m(W), m(S)) = -m(LP) - m(WR) - m(WW) + (n - 1)m(W) + m(S)$$

$$i_3 \cdot m_0 = i_3 \cdot m \implies m(LP) + m(WR) + m(WW) = (n-1)m(W) + m(S)$$

- If no process is writing then $m(WR) \le m(S)$.
- t_3 has concession if at least one process is waiting to read.

크

イロト イヨト イヨト イヨト

Proposition

The Readers-Writers net cannot deadlock (reach a marking where no transition has concession).

Proof.

If m(LP) + m(R) + m(W) > 0, it follows form the fact that t_1, t_2, t_5 or t_6 has concession.

If m(LP) + m(R) + m(W) = 0, it follows from i_1 and i_2 as they imply:

$$m(WR) + m(WW) = n$$

 $m(S) = n$

so t_3 or t_4 have concession.

Invariants for Coloured Petri Nets: Dining Philosophers

PH LEFT (x) + RIG HT (x) х PH (FORK LEFT(x) + RIGHT(x) colour PH = with ph1 | ph2 | ph3 | ph4 | ph5colour Fork = with $f1 \mid f2 \mid f3 \mid f4 \mid f5$ LEFT : $PH \rightarrow FORK$, RIGHT : $PH \rightarrow FORK$ var x : PHfun LEFT $x = case of ph1 \Rightarrow f2 \mid ph2 \Rightarrow f3 \mid ph3 \Rightarrow f4 \mid$ $ph4 \Rightarrow f5 \mid ph5 \Rightarrow f1$ fun RIGHT $x = case of ph1 \Rightarrow f1 \mid ph2 \Rightarrow f2 \mid ph3 \Rightarrow f3 \mid$ $ph4 \Rightarrow f4 \mid ph5 \Rightarrow f5$

Firing



Multisets (or Bags)

- A multiset *m*, over a non-empty and finite set *S* is a function $m: S \rightarrow \mathbb{N} = \{0, 1, 2, ...\}$
- m(s) is the number of appearances of s in m.
- notation: *M* is usually represented by:

$$\sum_{s\in S} m(s)s$$

$$S = \{a, b, c, d, e\},$$

 $m(a) = 3, m(b) = 1, m(c) = 0, m(d) = 183, m(e) = 4$
 $m = 3a + b + 4e + 183d$

- $s \in m \iff m(s) \neq 0$
- m(s) is a coefficient
- the empty multiset $m = \emptyset \iff m(s)$ for each $s \in S$.

Behaviours



Sequence:

(take forks, x = ph1)(take forks, x = ph3)(putdown forks, x = ph3)
Step-sequence:

{(take forks, x = ph1)(take forks, x = ph3)}{(putdown forks, x = ph3)} Partial order:



-∢ ≣ ▶

Invariants

• Invariants are equations that characterize all reachable markings.



- M(think) + M(eat) = ph1 + ph2 + ph3 + ph4 + ph5
 Each philosopher is either thinking or eating but not both. Also philosophers do not disappear and no new is born.
- $LEFT(M(eat)) + RIGHT(M(eat)) + M(free \ forks) = f_1 + f_2 + f_3 + f_4 + f_5$ where $LEFT(X) = \sum_{x \in X} LEFT(x)$, $RIGHT(X) = \sum_{x \in X} RIGHT(x)$ No philosopher can be eating at the same time as on of his neighbours.



(i1) M(think) + M(eat) = PH

(i2) LEFT(M(eat)) + RIGHT(M(eat)) + M(free forks) = FORK

Proposition

The above Coloured Petri net cannot deadlock.

Proof.

Assume that M is reachable from the initial marking. Then M satisfies (i1) and (i2). If $M(eat) \neq \emptyset$, i.e. $phj \in M(eat)$, then $(putdown \ fork, x = phj)$ can be fired. If $M(eat) = \emptyset$ it follows from (i1) and (i2) that M(think) = PH and $M(free \ forks) = FORK$

Then (*take forks*, x = phi), any $phi \in PM$ can be fired.

• Finding invariants can be reduced to finding non-negative integer solutions of some matrix equation:

$$W \cdot X = \mathbf{0}$$

where $\mathbf{0}$ is a vector of zeros,

W represents the structure of a net (incidence matrix), X represents an invariant.

- The number of invariants is infinite, but there is a finite number of linearly independent invariants
- Proper invariants are part of specification goals.
- Checking if an equation is an invariant is easy!

Writers Priority with Inhibitor Arcs



æ

Writers Priority with Selfloops



∢ ≣ ≯

E.

Writers Priority without Selfloops

